



POZNAN UNIVERSITY OF TECHNOLOGY

Wojciech Mioduszewski

Klasyfikacja danych opisanych za pomocą szeregów czasowych

Master's Thesis

Supervisor: dr inż. Jerzy Błaszczyński

Poznań, 2015

Contents

1	Wstęp	3
2	Background	5
2.1	Definicja szeregu czasowego	5
2.2	Przykładowe metody analizy danych czasowych	5
2.2.1	Regresja liniowa	5
2.2.2	Wyglądanie (“Smoothing”)	5
2.2.3	Modele ARIMA	6
2.2.4	Analiza spektrum	7
2.3	Inline formatting	7
2.4	Special characters	8
2.5	Figures	8
2.6	Tables	9
2.7	Source code examples.	10
2.8	Math	11
2.9	Algorithms	11
3	Concept and Design of the System	13
4	Implementation	15
5	Performance Evaluation	17
6	Conclusions	19
A	Users Guide	21
	Bibliography	23

Wstęp

Cel i zakres pracy

Cel: Opracowanie i implementacja różnych podejść do klasyfikacji danych czasowych.

Zadania:

- Zapoznać się z literaturą tematu.
- Opracować wybrane podejścia do klasyfikacji danych czasowych.
- Zaimplementować i udokumentować zaproponowane rozwiązania.
- Przeprowadzić eksperyment obliczeniowy

Początkowo celem niniejszej pracy była analiza szeregów czasowych zawierających dane ciśnienia w oku pacjentów zdrowych, oraz tych ze zdiagnozowaną jaskrą. Ponadto zamiarem było użycie do tego celu metody SAX, a następnie zbudowanie klasyfikatora potrafiącego sklasyfikować dane wytworzone przez tą metodę. Równie ważne było to, aby nie testować sposobów klasyfikacji tylko i wyłącznie na danych zebranych w celu oceny jaskry, lecz również sprawdzić jak wybrane i stworzone metody poradzą sobie w odniesieniu do innych szeregów czasowych.

The goal and the scope of the thesis

Celem pracy jest opracowanie / wykonanie analizy / zaprojektowanie /
Struktura pracy jest następująca. W rozdziale 2 przedstawiono przegląd literatury na temat Rozdział 3 jest poświęcony (kilka zdań). Rozdział 4 zawiera (kilka zdań) itd. Rozdział x stanowi podsumowanie pracy.

Background

2.1 Definicja szeregu czasowego

Szereg czasowy jest to seria pewnych obserwacji osadzonych w czasie. Można powiedzieć, że jest to przyporządkowanie danych liczbowych do odpowiadających im punktów w czasie, najczęściej z jednakowymi odstępami między kolejnymi wartościami.

<http://www.cs.put.poznan.pl/jstefanowski/aed/TPtimeseries.pdf>

2.2 Przykładowe metody analizy danych czasowych

Tak obszerny problem jak analiza danych czasowych musi nieść za sobą rozmaite metody przeprowadzania tej analizy. Poniżej przedstawiono kilka z nich.

2.2.1 Regresja liniowa

Regresja w odniesieniu do danych czasowych sprowadza się do estymowania liniowego trendu jaki prezentuje badany szereg[fav]. Koncepcyjnie metoda polega na stworzeniu funkcji liniowej, która w najbardziej dokładny sposób przybliży wartości na kolejnych obserwacjach. Matematyczny model regresji ma zatem następującą postać:

$$y = ax + b$$

Jako miarę błędu przyjmuje się sumę kwadratów różnicy między oszacowaniami, a wartościami właściwymi.

$$S = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

<http://www.cs.put.poznan.pl/jstefanowski/aed/TPDregresjawieloraka.pdf>

2.2.2 Wygładzanie (“Smoothing”)

Wygładzanie to metoda starająca się zniwelować ponadprzeciętne różnice wartości między kolejnymi pomiarami. Dzięki temu podejściu można dokładniej przyjrzeć się ogólnemu zarysowi funkcji, czy jej okresowymi trendami, rezygnując z precyzji. Podejście to pozwala

wyeliminować ewentualne szумы z badanego zbioru. Metoda ta jako argument przyjmuje szerokość okna k , w ramach którego będą uśredniane wartości. W pierwszym kroku liczy się średnią z k pierwszych wartości szeregu, następnie przesuwa okno o jeden element i znów liczy średnią z k wartości. Wyjściowy zbiór dla n -elementowego zbioru będzie miał $n - k$ wartości.

Poniżej dla porównania zaprezentowano wykres miary TFADJ w przeciągu doby 2.1, oraz jego wygładzony odpowiednik 2.2. Jak widać operacja wygładzania zredukowała skrajne wychylenia szeregu i wyklarowała paraboliczną tendencję tego zbioru.

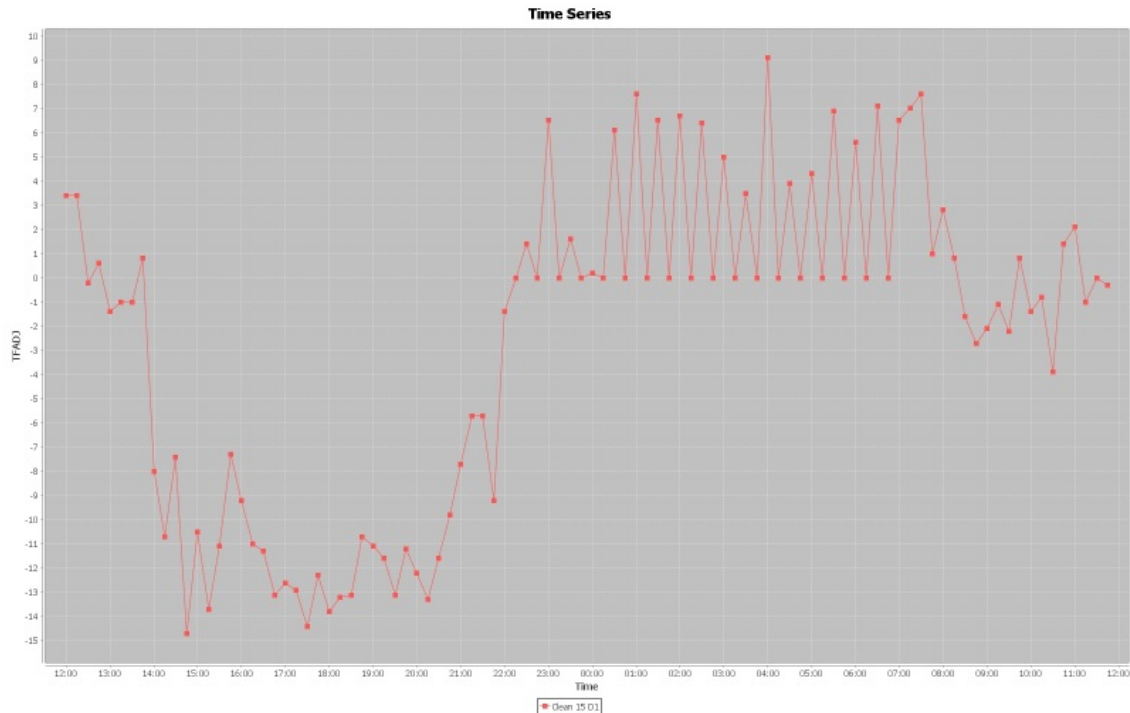


Figure 2.1: Czysty sygnał TFADJ

2.2.3 Modele ARIMA

Model ARIMA (ang. Autoregressive Integrated Moving Average) koncepcyjnie składają się z trzech części - jest to autoregresja, zintegrowanie oraz średnia ruchoma.

Autoregresja jest to idea, która skupia się na wyrażeniu bieżącej wartości na podstawie poprzednich. Funkcja dla $n = 2$ wygląda w ten sposób:

$$x_t = \omega_{t-2}x_{t-2} - \omega_{t-1}x_{t-1} + \omega_t$$

Średnia ruchoma ideowo jest bardzo zbliżona do autoregresji, jednak skupia się na zaburzeniach (ang. lags) w szeregu, a nie bezpośrednio na wartościach. Wzór przedstawia się analogicznie jak w autoregresji.

Integracja z kolei pozwala na zastosowanie modelu ARIMA do procesów niestacjonarnych, które da się sprowadzić do procesów stacjonarnych za pomocą różnic (ang. difference equations) pomiędzy wartością obecną, a poprzednią.

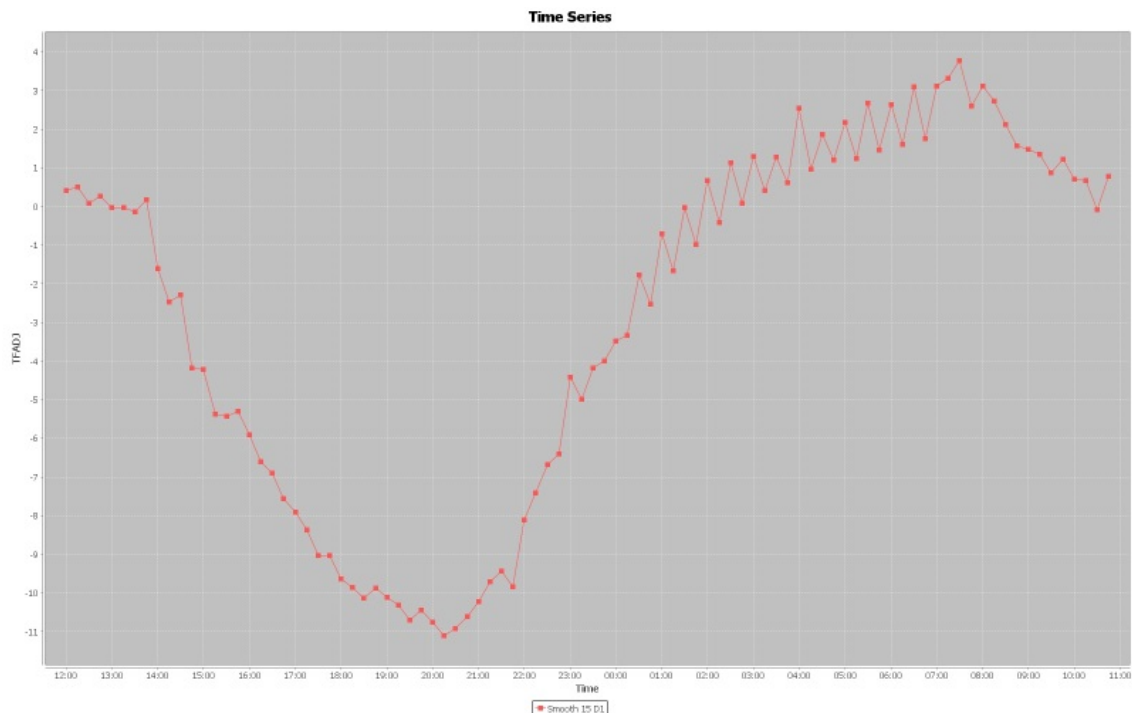


Figure 2.2: Wygładzony sygnał TFADJ

Podsumowując, modele ARIMA doskonale spisują się w prognozowaniu wartości, bazując na danych historycznych szeregu, jednak nie są najlepszym wyborem jako metoda klasyfikacji. Ponadto metoda nie jest najłatwiejsza matematycznie. Biorąc pod uwagę te argumenty, metoda ta została odrzucona.

2.2.4 Analiza spektrum

Paragraph

Subparagraph

2.3 Inline formatting

We suggest using *Insets*, like:

strong for strong emphasizing some text.

emph for emphasizing some text.

Code for formatting of names of modules, procedures, class names, variables, etc.

path for formatting of file names and directories, like `/usr/share/doc/packages/texlive-latex`. The names are properly broken at the ends of lines. However, path names containing special \LaTeX characters must be typeset using ERT and the `\dcspath` command, e.g. `sample_file`.

kbd for formatting of shortcuts, e.g.: `Ctrl-c`.

cmd for formatting system commands.
name for formatting other special names.

2.4 Special characters

1. Non-breaking space can be inserted using **Ctrl-space**. It produces “~” in \LaTeX code.
2. A normal, inter-word space can be inserted using **Ctrl-Alt-space**. It produces “\ ” in \LaTeX code. This type of space is useful for formatting spacing after dots, e.g. here. By default \LaTeX produces here a longer space used for separating whole sentences.
3. A thin space can be produced by **Ctrl-Shift-space**, e.g. here. It produces “\,” in \LaTeX code.
4. Sentence-ending space can be inserted using **Ctrl-.**, which produces “\@.” in \LaTeX code. This type of space is useful in sentences ending with a capital letter. In such cases \LaTeX recognizes the last word as a acronym and places a regular inter-word space instead of inter-sentence space. Consider the following example:

This can be achieved by using HTTP. This protocol...

5. Hyphenation indicator can be inserted using **Ctrl- -**, which is used for marking possible places of hyphenation, e.g. democracy.

2.5 Figures

The figures should be put in floats, like Fig. 2.3. You can also reference figures using **prettyref** package like this: Fig. 2.3.



Figure 2.3: Example figure

It is possible to combine several pictures inside one float. Just insert a float inside a float. See Fig. 2.4 for example. Please note the horizontal spacing between subfigures.



(a) The first subfigure



(b) The second subfigure

Figure 2.4: Example figure

2.6 Tables

Tables should have captions above like Table 2.1. Use small sans-serif fonts inside tables.

Table 2.1: Example table

Column 1	Column 2	Column 3
One	1	4
Two	2	5
Three	3	6

2.7 Source code examples

There are a few different methods of including sample codes:

1. Using standard **LyX-Code** style:

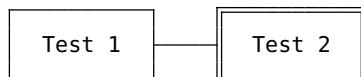
```
#include <stdio.h>

int main() {
    printf("Hello world!\n");
    return 0;
}
```

Note 1: Empty lines must contain at least one single space to remain visible.

Note 2: There is no way to activate automatic syntax highlighting inside **LyX-Code**. However, you can use normal inline formatting inside.

Note 3: **Lyx-Code** can contain special characters, so it can be used to produce some ASCII art, e.g.:



2. By inserting *Program Listing*:

```
#include <stdio.h>

int main() {
    printf("Hello world!\n");
    return 0;
}
```

Note: By default the **lstlisting** environment does not add any left margin. You can change it by adding **xleftmargin** in the *Settings* ▸ *Advanced* dialog box, e.g.:

```
procedure sayHello()
```

3. By inserting **L^AT_EX** Code (ERT block) and using **codeblock** environment:

```
#include <stdio.h>

int main() {
    printf("Hello world!\n");
    return 0;
}
```

4. The **listings** package can produce floats by itself. See Listing. 2.1 for example.
5. And finally, You can include code from external file:

```
\documentclass[11pt,a4paper,polish,thesis]{dcsbook}

\usepackage[utf8]{inputenc}
\usepackage{babel}
\setcounter{secnumdepth}{4}
```

Listing 2.1: The Hello World program in C

```
#include <stdio.h>

int main() {
    printf("Hello world!\n");
    return 0;
}
```

```
\setcounter{tocdepth}{3}
```

```
\begin{document}
```

2.8 Math

Can be put inline like this: $S = \sum_{i=1}^{i=K} x_i^2$ or in dedicated lines:

$$S = \sum_{i=1}^{i=K} x_i^2$$

The equations can be also numbered like equation [2.1](#).

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (2.1)$$

2.9 Algorithms

Use `dcsalg` package or directly `algorithmicx` package.

Concept and Design of the System

Implementation

Performance Evaluation

Conclusions

Appendix A

Users Guide

Bibliography